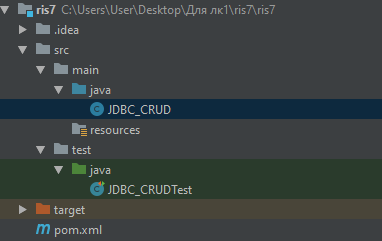
**Задание №6 Тестирование приложений с применением JUnit**

Для теста и проверки работы библиотек рассмотрим пример:



Для начала добавим базу данных и разберем тесты:

-- --------------------------------------------------------

-- Хост: 127.0.0.1

-- Версия сервера: 8.0.15 - MySQL Community Server - GPL

-- Операционная система: Win64

-- HeidiSQL Версия: 10.1.0.5464

-- --------------------------------------------------------

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET NAMES utf8 \*/;

/\*!50503 SET NAMES utf8mb4 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

-- Дамп структуры базы данных jdbctest

CREATE DATABASE IF NOT EXISTS `jdbctest` /\*!40100 DEFAULT CHARACTER SET cp1251 \*/;

USE `jdbctest`;

-- Дамп структуры для таблица jdbctest.programmers

CREATE TABLE IF NOT EXISTS `programmers` (

`Programmer\_id` int(11) NOT NULL AUTO\_INCREMENT,

`Name` varchar(50) DEFAULT NULL,

`Location` varchar(50) DEFAULT NULL,

`Qualification` varchar(50) DEFAULT NULL,

PRIMARY KEY (`Programmer\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=cp1251;

-- Экспортируемые данные не выделены.

/\*!40101 SET SQL\_MODE=IFNULL(@OLD\_SQL\_MODE, '') \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=IF(@OLD\_FOREIGN\_KEY\_CHECKS IS NULL, 1, @OLD\_FOREIGN\_KEY\_CHECKS) \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

Не забываем про pom, в нем опишем библиотеки для нашего ПО:

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>ris7</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-api</artifactId>  
 <version>5.8.0</version>  
 <scope>test</scope>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.23</version>  
 </dependency>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 <scope>test</scope>  
 </dependency>  
  
 </dependencies>  
 <properties>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 </properties>  
  
</project>

Добавим основной класс с методом:

import java.sql.\*;  
  
*/\*\*  
 \* Simple CRUD class for JDBC learning purposes.  
 \* Using DriverManager to get connection to database.  
 \* Table columns: Programmer\_id, Name, Location, Qualification.  
 \*/*public class JDBC\_CRUD {  
 private Connection connection;  
 private String tableName;  
   
 */\*\*  
 \* Constructor creates connection and statement.  
 \** ***@param*** *adress Adress of database.  
 \** ***@param*** *login Login for database.  
 \** ***@param*** *password Password for database.  
 \** ***@throws*** *SQLException  
 \*/* public JDBC\_CRUD(String adress, String login, String password, String tableName) throws SQLException {  
 connection=DriverManager.*getConnection*(adress,login,password);  
 this.tableName=tableName;  
 }  
   
 */\*\*  
 \* Creates new programmer in database.  
 \** ***@param*** *name Programmer name.  
 \** ***@param*** *location Programmer location.  
 \** ***@param*** *qualification Programmer Qualification.  
 \** ***@throws*** *SQLException  
 \*/* public void createProgrammer(int id, String name, String location, String qualification) throws SQLException {  
 String query = "INSERT INTO "+tableName+" (Programmer\_id, Name, Location, Qualification) VALUES (?,?,?,?);";  
 PreparedStatement pstmt = connection.prepareStatement(query);  
 pstmt.setInt(1, id);  
 pstmt.setString(2, name);  
 pstmt.setString(3, location);  
 pstmt.setString(4, qualification);  
 pstmt.executeUpdate();  
 }  
   
 */\*\*  
 \* Delete programmer from database.  
 \** ***@param*** *\** ***@throws*** *SQLException  
 \*/* public void deleteProgrammer(int id) throws SQLException {  
 String query = "DELETE FROM "+tableName+" WHERE Programmer\_id=?;";  
 PreparedStatement pstmt = connection.prepareStatement(query);  
 pstmt.setInt(1, id);  
 pstmt.executeUpdate();  
 }  
   
 */\*\*  
 \* Show table and return ResultSet.  
 \** ***@return*** *Returns table contents in ResultSet.  
 \** ***@throws*** *SQLException  
 \*/* public ResultSet readTable() throws SQLException {  
 Statement stmt = connection.createStatement();  
 ResultSet rs = stmt.executeQuery("SELECT \* FROM "+tableName);  
 int count=rs.getMetaData().getColumnCount();  
 while(rs.next()) {  
 for(int i=0; i<count; i++) {  
 System.*out*.print(rs.getString(i+1)+" ");  
 }  
 System.*out*.print("\n");  
 }  
 return rs;  
 }  
   
 */\*\*  
 \* Updates record based on ID.  
 \** ***@param*** *id Programmer\_id.  
 \** ***@param*** *name Programmer Name.  
 \** ***@param*** *location Programmer location  
 \** ***@param*** *qualification Programmer qualification  
 \** ***@throws*** *SQLException  
 \*/* public void updateProgrammer(int id, String name, String location, String qualification) throws SQLException {  
 String query = "UPDATE "+tableName+" SET Name=?, Location=?, Qualification=? WHERE Programmer\_id=?;";  
 PreparedStatement pstmt = connection.prepareStatement(query);  
 pstmt.setString(1, name);  
 pstmt.setString(2, location);  
 pstmt.setString(3, qualification);  
 pstmt.setInt(4, id);  
 pstmt.executeUpdate();  
 }  
   
 */\*\*  
 \* Truncates table.  
 \** ***@throws*** *SQLException  
 \*/* public void clear() throws SQLException {  
 Statement stmt = connection.createStatement();  
 stmt.executeUpdate("TRUNCATE TABLE "+tableName);  
 }  
   
}

Еще раз рассмотрим основные методы класса.

Осталось подключить Junit и осуществить тестирование наших методов:

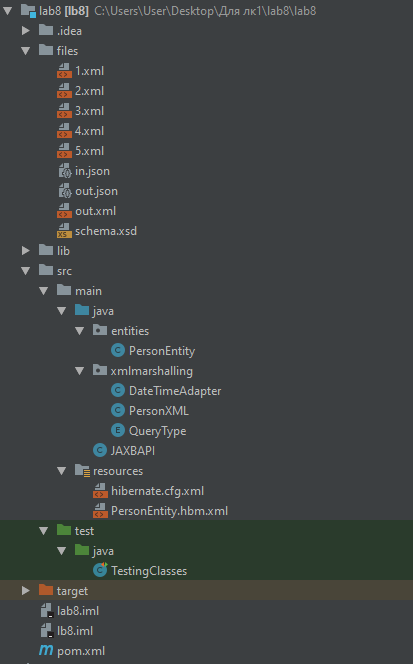
import org.junit.Before;  
import org.junit.Ignore;  
import org.junit.jupiter.api.Test;  
  
import java.sql.ResultSet;  
import java.sql.SQLException;  
  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class JDBC\_CRUDTest {  
 private static final String *ADRESS* = "jdbc:mysql://localhost:3306/jdbctest";  
 private static final String *LOGIN*="root";  
 private static final String *PASSWORD*="root";  
 private static final String *TABLE\_NAME*="programmers";  
  
 @Test  
 void testClass() {  
 try {  
 new JDBC\_CRUD(*ADRESS*,*LOGIN*,*PASSWORD*,*TABLE\_NAME*);  
 }  
 catch(SQLException e) {  
 e.printStackTrace();  
 *fail*("Failed to connect. SQLException.");  
 }  
 }  
  
  
 @Test  
 void testCreate() {  
 try {  
 JDBC\_CRUD db = new JDBC\_CRUD(*ADRESS*,*LOGIN*,*PASSWORD*,*TABLE\_NAME*);  
 db.createProgrammer(1, "Vasya", "St.Petersburg", "Junior");  
 db.createProgrammer(2, "Maksim", "St.Petersburg", "Middle");  
 db.createProgrammer(3, "Vova", "Moscow", "Senior");  
 } catch (SQLException e) {  
 e.printStackTrace();  
 *fail*("Failed to create new programmer. SQLException");  
 }  
 }  
  
 @Test  
 void testDelete() {  
 try {  
 JDBC\_CRUD db = new JDBC\_CRUD(*ADRESS*,*LOGIN*,*PASSWORD*,*TABLE\_NAME*);  
 db.deleteProgrammer(1);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 *fail*("Failed to delete programmer. SQLException");  
 }  
 }  
   
 @Test  
 void testRead() {  
 ResultSet rs=null;  
 try {  
 JDBC\_CRUD db = new JDBC\_CRUD(*ADRESS*,*LOGIN*,*PASSWORD*,*TABLE\_NAME*);  
 rs=db.readTable();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 *fail*("Failed to read database. SQLException");  
 }  
 *assertNotNull*(rs);  
 }  
  
 @Ignore  
 @Test  
 void truncateTable() {  
 try {  
 JDBC\_CRUD db = new JDBC\_CRUD(*ADRESS*,*LOGIN*,*PASSWORD*,*TABLE\_NAME*);  
 db.clear();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 *fail*("Failed to truncate table. SQLException");  
 }  
 }  
   
 @Test  
 void testUpdate() {  
 try {  
 JDBC\_CRUD db = new JDBC\_CRUD(*ADRESS*,*LOGIN*,*PASSWORD*,*TABLE\_NAME*);  
 db.updateProgrammer(2, "Petya", "St.Petersburg", "Junior");  
 } catch (SQLException e) {  
 e.printStackTrace();  
 *fail*("Failed to update database. SQLException");  
 }  
 }  
}

Обратите внимание на содержание в методах – только бизнес-логика, никаких данных. Все данные приходят со стороны тестов. Как и откуда данные поступают в тесты – дело второе (можно из совершенно любых источников их подгружать.).

Следующую часть разберем на примере, с применением подхода TDD. В свою очередь, данный пример мы рассматриваем довольно нишевый, как бы «в вакууме», потому что сложно предугадать, какую часть системы нужно будет разработать именно нам. В примере почти вся логика вынесена в тесты – чего в реальной реализации может и не быть.

Разработка через TDD обычно покрывает больший процент времени, имеет большую сложность, однако нивелирует основные ошибки, которые можно допустить на этапе компиляции при стандартной разработке. Порой экономически не целесообразно прибегать к данному способу или подходку к реализации ПО.

Рассмотрим структуру проекта:



Какие части системы мы тут видим? В первую очередь обратите внимание, что у нас нет точки входа для запуска ПО и меню пользователя. Так же данные поступают непосредственно их внешнего источника – xml, json.

Начнем с конфигурации проекта, для этого понадобится pom. Часть кода пом автоконфигурации – можете не копировать весь.:

<?xml version="1.0" encoding="UTF-8"?>  
  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.bsuir.springboot</groupId>  
 <artifactId>lb8</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <!-- Inherit defaults from Spring Boot -->  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.2.4.RELEASE</version>  
 </parent>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <maven.compiler.source>1.7</maven.compiler.source>  
 <maven.compiler.target>1.7</maven.compiler.target>  
 </properties>  
  
 <dependencies>  
 <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-api</artifactId>  
 <version>5.6.0</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.hibernate</groupId>  
 <artifactId>hibernate-agroal</artifactId>  
 <version>5.4.12.Final</version>  
 <type>pom</type>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.19</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->  
 <dependency>  
 <groupId>com.google.code.gson</groupId>  
 <artifactId>gson</artifactId>  
 <version>2.8.6</version>  
 </dependency>  
 </dependencies>  
 <build>  
 <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->  
 <plugins>  
 <!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean\_Lifecycle -->  
 <plugin>  
 <artifactId>maven-clean-plugin</artifactId>  
 <version>3.1.0</version>  
 </plugin>  
 <!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_jar\_packaging -->  
 <plugin>  
 <artifactId>maven-resources-plugin</artifactId>  
 <version>3.0.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.8.0</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.22.1</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-jar-plugin</artifactId>  
 <version>3.0.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-install-plugin</artifactId>  
 <version>2.5.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-deploy-plugin</artifactId>  
 <version>2.8.2</version>  
 </plugin>  
 <!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site\_Lifecycle -->  
 <plugin>  
 <artifactId>maven-site-plugin</artifactId>  
 <version>3.7.1</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-project-info-reports-plugin</artifactId>  
 <version>3.0.0</version>  
 </plugin>  
 </plugins>  
 </pluginManagement>  
 </build>  
</project>

Класс Entity (POJO, Entity, Bean, dto)

package entities;  
  
import javax.persistence.\*;  
import java.sql.Date;  
import java.util.Objects;  
  
@Entity  
@Table(name = "person", schema = "lb8")  
@Inheritance(strategy = InheritanceType.*SINGLE\_TABLE*)  
public class PersonEntity {  
 private String id;  
 private String surname;  
 private String forename;  
 private String patronymic;  
 private Date dob;  
 private byte sex;  
 private String passportSeries;  
 private String passportNumber;  
 private String city;  
 private String address;  
 private String phoneHome;  
 private String phoneMobile;  
 private String citizenship;  
 private byte dutyBound;  
  
 @Id  
 @Column(name = "id", nullable = false, length = 45)  
 public String getId() {  
 return id;  
 }  
  
 public void setId(String id) {  
 this.id = id;  
 }  
  
 @Basic  
 @Column(name = "surname", nullable = false, length = 45)  
 public String getSurname() {  
 return surname;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 @Basic  
 @Column(name = "forename", nullable = false, length = 45)  
 public String getForename() {  
 return forename;  
 }  
  
 public void setForename(String forename) {  
 this.forename = forename;  
 }  
  
 @Basic  
 @Column(name = "patronymic", nullable = false, length = 45)  
 public String getPatronymic() {  
 return patronymic;  
 }  
  
 public void setPatronymic(String patronymic) {  
 this.patronymic = patronymic;  
 }  
  
 @Basic  
 @Column(name = "DOB", nullable = false)  
 public Date getDob() {  
 return dob;  
 }  
  
 public void setDob(Date dob) {  
 this.dob = dob;  
 }  
  
 @Basic  
 @Column(name = "sex", nullable = false)  
 public byte getSex() {  
 return sex;  
 }  
  
 public void setSex(byte sex) {  
 this.sex = sex;  
 }  
  
 @Basic  
 @Column(name = "passport\_series", nullable = false, length = 45)  
 public String getPassportSeries() {  
 return passportSeries;  
 }  
  
 public void setPassportSeries(String passportSeries) {  
 this.passportSeries = passportSeries;  
 }  
  
 @Basic  
 @Column(name = "passport\_number", nullable = false, length = 45)  
 public String getPassportNumber() {  
 return passportNumber;  
 }  
  
 public void setPassportNumber(String passportNumber) {  
 this.passportNumber = passportNumber;  
 }  
  
 @Basic  
 @Column(name = "city", nullable = false, length = 45)  
 public String getCity() {  
 return city;  
 }  
  
 public void setCity(String city) {  
 this.city = city;  
 }  
  
 @Basic  
 @Column(name = "address", nullable = false, length = 45)  
 public String getAddress() {  
 return address;  
 }  
  
 public void setAddress(String address) {  
 this.address = address;  
 }  
  
 @Basic  
 @Column(name = "phone\_home", length = 45)  
 public String getPhoneHome() {  
 return phoneHome;  
 }  
  
 public void setPhoneHome(String phoneHome) {  
 this.phoneHome = phoneHome;  
 }  
  
 @Basic  
 @Column(name = "phone\_mobile", length = 45)  
 public String getPhoneMobile() {  
 return phoneMobile;  
 }  
  
 public void setPhoneMobile(String phoneMobile) {  
 this.phoneMobile = phoneMobile;  
 }  
  
 @Basic  
 @Column(name = "citizenship", nullable = false, length = 45)  
 public String getCitizenship() {  
 return citizenship;  
 }  
  
 public void setCitizenship(String citizenship) {  
 this.citizenship = citizenship;  
 }  
  
 @Basic  
 @Column(name = "duty\_bound", nullable = false)  
 public byte getDutyBound() {  
 return dutyBound;  
 }  
  
 public void setDutyBound(byte dutyBound) {  
 this.dutyBound = dutyBound;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
  
 PersonEntity that = (PersonEntity) o;  
  
 if (sex != that.sex) return false;  
 if (!Objects.*equals*(id, that.id)) return false;  
 if (!Objects.*equals*(surname, that.surname)) return false;  
 if (!Objects.*equals*(forename, that.forename)) return false;  
 if (!Objects.*equals*(patronymic, that.patronymic)) return false;  
 if (!Objects.*equals*(dob, that.dob)) return false;  
 if (!Objects.*equals*(passportSeries, that.passportSeries))  
 return false;  
 if (!Objects.*equals*(passportNumber, that.passportNumber))  
 return false;  
 if (!Objects.*equals*(city, that.city)) return false;  
 if (!Objects.*equals*(address, that.address)) return false;  
 if (!Objects.*equals*(citizenship, that.citizenship)) return false;  
 return dutyBound == that.dutyBound;  
 }  
  
 @Override  
 public int hashCode() {  
 int result = id != null ? id.hashCode() : 0;  
 result = 31 \* result + (surname != null ? surname.hashCode() : 0);  
 result = 31 \* result + (forename != null ? forename.hashCode() : 0);  
 result = 31 \* result + (patronymic != null ? patronymic.hashCode() : 0);  
 result = 31 \* result + (dob != null ? dob.hashCode() : 0);  
 result = 31 \* result + (int) sex;  
 result = 31 \* result + (passportSeries != null ? passportSeries.hashCode() : 0);  
 result = 31 \* result + (passportNumber != null ? passportNumber.hashCode() : 0);  
 result = 31 \* result + (city != null ? city.hashCode() : 0);  
 result = 31 \* result + (address != null ? address.hashCode() : 0);  
 result = 31 \* result + (citizenship != null ? citizenship.hashCode() : 0);  
 result = 31 \* result + (int) dutyBound;  
 return result;  
 }  
}

Для маппинга можете использовать любой подход. Для примера тут есть заготовка через xml, однако все присутствует и в виде аннотаций:

<?xml version='1.0' encoding='utf-8'?>  
<!DOCTYPE hibernate-mapping PUBLIC  
 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  
<hibernate-mapping>  
  
 <class name="entities.PersonEntity" table="person" schema="lb8">  
 <id name="id">  
 <column name="id" sql-type="varchar(45)" length="45"/>  
 </id>  
 <property name="surname">  
 <column name="surname" sql-type="varchar(45)" length="45"/>  
 </property>  
 <property name="forename">  
 <column name="forename" sql-type="varchar(45)" length="45"/>  
 </property>  
 <property name="patronymic">  
 <column name="patronymic" sql-type="varchar(45)" length="45"/>  
 </property>  
 <property name="dob">  
 <column name="DOB" sql-type="date"/>  
 </property>  
 <property name="sex">  
 <column name="sex" sql-type="tinyint"/>  
 </property>  
 <property name="passportSeries">  
 <column name="passport\_series" sql-type="varchar(45)" length="45"/>  
 </property>  
 <property name="passportNumber">  
 <column name="passport\_number" sql-type="varchar(45)" length="45"/>  
 </property>  
 <property name="city">  
 <column name="city" sql-type="varchar(45)" length="45"/>  
 </property>  
 <property name="address">  
 <column name="address" sql-type="varchar(45)" length="45"/>  
 </property>  
 <property name="phoneHome">  
 <column name="phone\_home" sql-type="varchar(45)" length="45"/>  
 </property>  
 <property name="phoneMobile">  
 <column name="phone\_mobile" sql-type="varchar(45)" length="45"/>  
 </property>  
 <property name="citizenship">  
 <column name="citizenship" sql-type="varchar(45)" length="45"/>  
 </property>  
 <property name="dutyBound">  
 <column name="duty\_bound" sql-type="tinyint"/>  
 </property>  
 </class>  
</hibernate-mapping>

Так же нам необходима конфигурация:

<?xml version='1.0' encoding='utf-8'?>  
<!DOCTYPE hibernate-configuration PUBLIC  
 "-//Hibernate/Hibernate Configuration DTD//EN"  
 "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  
<hibernate-configuration>  
 <session-factory>  
 <!-- <property name="hibernate.hbm2ddl.auto">update</property>-->  
 <property name="connection.url">jdbc:mysql://localhost:3306/lb8?useUnicode=true&amp;serverTimezone=UTC</property>  
 <property name="connection.driver\_class">com.mysql.cj.jdbc.Driver</property>  
 <property name="connection.username">root</property>  
 <property name="connection.password">root</property>  
 <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>  
 <mapping class="entities.PersonEntity"/>  
 <mapping resource="PersonEntity.hbm.xml"/>  
 </session-factory>  
</hibernate-configuration>

Так же нам потребуется класс маршалинга для фалов XML:

package xmlmarshalling;  
import entities.PersonEntity;  
  
import javax.xml.bind.annotation.XmlAccessType;  
import javax.xml.bind.annotation.XmlAccessorType;  
import javax.xml.bind.annotation.XmlRootElement;  
import javax.xml.bind.annotation.XmlType;  
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;  
import java.sql.Date;  
  
  
@XmlType(name = "person")  
@XmlAccessorType(XmlAccessType.*FIELD*)  
@XmlRootElement  
public class PersonXML {  
 private String id;  
 private String surname;  
 private String forename;  
 private String patronymic;  
 @XmlJavaTypeAdapter(DateTimeAdapter.class)  
 private Date dob;  
 private byte sex;  
 private String passportSeries;  
 private String passportNumber;  
 private String city;  
 private String address;  
 private String phoneHome;  
 private String phoneMobile;  
 private String citizenship;  
 private byte dutyBound;  
  
 private QueryType qtype;  
  
 public PersonXML() {  
 }  
  
 public PersonXML(PersonEntity origin) {  
 this.id = origin.getId();  
 this.surname = origin.getSurname();  
 this.forename = origin.getForename();  
 this.patronymic = origin.getPatronymic();  
 this.dob = origin.getDob();  
 this.sex = origin.getSex();  
 this.passportSeries = origin.getPassportSeries();  
 this.passportNumber = origin.getPassportNumber();  
 this.city = origin.getCity();  
 this.address = origin.getAddress();  
 this.phoneHome = origin.getPhoneHome();  
 this.phoneMobile = origin.getPhoneMobile();  
 this.citizenship = origin.getCitizenship();  
 this.dutyBound = origin.getDutyBound();  
 this.qtype = null;  
 }  
  
 public PersonEntity getEntity() {  
 PersonEntity emp = new PersonEntity();  
 emp.setId(id);  
 emp.setSurname(surname);  
 emp.setForename(forename);  
 emp.setPatronymic(patronymic);  
 emp.setDob(dob);  
 emp.setSex(sex);  
 emp.setPassportSeries(passportSeries);  
 emp.setPassportNumber(passportNumber);  
 emp.setCity(city);  
 emp.setAddress(address);  
 emp.setPhoneHome(phoneHome);  
 emp.setPhoneMobile(phoneMobile);  
 emp.setCitizenship(citizenship);  
 emp.setDutyBound(dutyBound);  
 return emp;  
 }  
  
 public String getId() {  
 return id;  
 }  
  
 public void setId(String id) {  
 this.id = id;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 public String getForename() {  
 return forename;  
 }  
  
 public void setForename(String forename) {  
 this.forename = forename;  
 }  
  
 public String getPatronymic() {  
 return patronymic;  
 }  
  
 public void setPatronymic(String patronymic) {  
 this.patronymic = patronymic;  
 }  
  
 public Date getDob() {  
 return dob;  
 }  
  
 public void setDob(Date dob) {  
 this.dob = dob;  
 }  
  
 public byte getSex() {  
 return sex;  
 }  
  
 public void setSex(byte sex) {  
 this.sex = sex;  
 }  
  
 public String getPassportSeries() {  
 return passportSeries;  
 }  
  
 public void setPassportSeries(String passportSeries) {  
 this.passportSeries = passportSeries;  
 }  
  
 public String getPassportNumber() {  
 return passportNumber;  
 }  
  
 public void setPassportNumber(String passportNumber) {  
 this.passportNumber = passportNumber;  
 }  
  
 public String getCity() {  
 return city;  
 }  
  
 public void setCity(String city) {  
 this.city = city;  
 }  
  
 public String getAddress() {  
 return address;  
 }  
  
 public void setAddress(String address) {  
 this.address = address;  
 }  
  
 public String getPhoneHome() {  
 return phoneHome;  
 }  
  
 public void setPhoneHome(String phoneHome) {  
 this.phoneHome = phoneHome;  
 }  
  
 public String getPhoneMobile() {  
 return phoneMobile;  
 }  
  
 public void setPhoneMobile(String phoneMobile) {  
 this.phoneMobile = phoneMobile;  
 }  
  
 public String getCitizenship() {  
 return citizenship;  
 }  
  
 public void setCitizenship(String citizenship) {  
 this.citizenship = citizenship;  
 }  
  
 public byte getDutyBound() {  
 return dutyBound;  
 }  
  
 public void setDutyBound(byte dutyBound) {  
 this.dutyBound = dutyBound;  
 }  
  
 public QueryType getQtype() {  
 return qtype;  
 }  
  
 public void setQtype(QueryType qtype) {  
 this.qtype = qtype;  
 }  
}

Далее разберем Перечисление, которое нам понадобится для определения действий и операций. Каждое из них хранится в данных XML, JSON.

package xmlmarshalling;  
  
public enum QueryType {  
 *CREATE*,  
 *READ*,  
 *UPDATE*,  
 *DELETE*}

Т.к. мы работать с датой будем, то потребуется класс, способствующий оборачивать данные в объект даты:

package xmlmarshalling;  
  
import javax.xml.bind.annotation.adapters.XmlAdapter;  
import java.sql.Date;  
import java.text.SimpleDateFormat;  
  
class DateTimeAdapter extends XmlAdapter<String, Date> {  
 @Override  
 public Date unmarshal(String xml) {  
 return Date.*valueOf*(xml);  
 }  
  
 @Override  
 public String marshal(Date object) {  
 return new SimpleDateFormat("yyyy-MM-dd").format(object);  
 }  
}

На данном этапе мы должны описать тесты для будущей логики приложения:

import entities.PersonEntity;  
import xmlmarshalling.PersonXML;  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
import org.hibernate.Session;  
  
import javax.xml.bind.JAXBContext;  
import javax.xml.bind.JAXBException;  
import javax.xml.bind.Unmarshaller;  
import java.io.\*;  
import java.nio.charset.StandardCharsets;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.util.ArrayList;  
  
class JAXBAPI {  
  
 static PersonEntity executeXML(String filename, Session session) throws IOException, JAXBException {  
 PersonXML personXML = *getQuery*(filename);  
 switch (personXML.getQtype()) {  
 case *CREATE*:  
 *create*(personXML.getEntity(), session);  
 break;  
 case *UPDATE*:  
 *update*(personXML.getEntity(), session);  
 break;  
 case *READ*:  
 return *read*(personXML.getEntity(), session);  
 case *DELETE*:  
 *delete*(personXML.getEntity(), session);  
 break;  
 }  
 return null;  
 }  
  
 static PersonXML getQuery(String filename) throws IOException, JAXBException {  
 StringBuilder strbldr = new StringBuilder();  
 ArrayList<String> lines = (ArrayList<String>) Files.*readAllLines*(Paths.*get*(filename), StandardCharsets.*UTF\_8*);  
 for (String line : lines) {  
 strbldr.append(line);  
 }  
 JAXBContext jaxbContext = JAXBContext.*newInstance*(PersonXML.class);  
 Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();  
 return (PersonXML) jaxbUnmarshaller.unmarshal(new StringReader(strbldr.toString()));  
 }  
  
 private static void create(PersonEntity entity, Session session) {  
 session.beginTransaction();  
 session.save(entity);  
 session.getTransaction().commit();  
 }  
  
 private static void update(PersonEntity entity, Session session) {  
 session.beginTransaction();  
 PersonEntity mod = session.get(PersonEntity.class, entity.getId());  
 if (entity.getSurname() != null) {  
 mod.setSurname(entity.getSurname());  
 }  
 if (entity.getAddress() != null) {  
 mod.setAddress(entity.getAddress());  
 }  
 mod.setDutyBound(entity.getDutyBound());  
 session.update(mod);  
 session.getTransaction().commit();  
 }  
  
 private static PersonEntity read(PersonEntity entity, Session session) {  
 return session.get(PersonEntity.class, entity.getId());  
 }  
  
 private static void delete(PersonEntity entity, Session session) {  
 session.beginTransaction();  
 PersonEntity myObject = session.load(PersonEntity.class, entity.getId());  
 session.delete(myObject);  
 session.getTransaction().commit();  
 }  
  
 static void addFromJson(@SuppressWarnings("SameParameterValue") String filename, Session session) throws IOException {  
 Gson gson = new GsonBuilder().setPrettyPrinting().create();  
 try (Reader reader = new FileReader(filename)) {  
 PersonEntity personEntity = gson.fromJson(reader, PersonEntity.class);  
 *create*(personEntity, session);  
 }  
 }  
  
 static void dumpToJson(@SuppressWarnings("SameParameterValue") String filename, Session session, @SuppressWarnings("SameParameterValue") String ID) throws IOException {  
 Gson gson = new GsonBuilder().setPrettyPrinting().create();  
 PersonEntity personEntity = session.find(PersonEntity.class, ID);  
 try (FileWriter writer = new FileWriter(filename)) {  
 gson.toJson(personEntity, writer);  
 }  
 }  
}

Далее класс – основной обработчик, содержащий всю бизнес-логику приложения без данных, пользовательского меню и пр. Т.е. только логика.

import entities.PersonEntity;  
import xmlmarshalling.PersonXML;  
import org.hibernate.HibernateException;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
import org.junit.jupiter.api.\*;  
  
import javax.xml.bind.JAXBContext;  
import javax.xml.bind.JAXBException;  
import javax.xml.bind.Marshaller;  
import java.io.\*;  
import java.util.Objects;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)  
class TestingClasses {  
  
 private static SessionFactory *sessionFactory* = null;  
  
 @BeforeAll  
 static void configure() {  
 try {  
 Configuration configuration = new Configuration();  
 configuration.configure();  
 *sessionFactory* = configuration.buildSessionFactory();  
 } catch (Throwable ex) {  
 throw new ExceptionInInitializerError(ex);  
 }  
 }  
  
 private static Session getSession() throws HibernateException {  
 return *sessionFactory*.openSession();  
 }  
  
 @Test  
 @Order(1)  
 void creation() {  
 try {  
 Session session = *getSession*();  
 JAXBAPI.*executeXML*("files/1.xml", session);  
 PersonEntity personEntity = JAXBAPI.*executeXML*("files/2.xml", session);  
 PersonXML personXML = JAXBAPI.*getQuery*("files/1.xml");  
 *assertEquals*(personEntity, personXML.getEntity());  
 } catch (IOException | JAXBException e) {  
 *fail*();  
 }  
 }  
  
 @Test  
 @Order(2)  
 void update() {  
 try {  
 Session session = *getSession*();  
 int prev = Objects.*requireNonNull*(JAXBAPI.*executeXML*("files/2.xml", session)).getDutyBound();  
 JAXBAPI.*executeXML*("files/3.xml", session);  
 int post = Objects.*requireNonNull*(JAXBAPI.*executeXML*("files/2.xml", session)).getDutyBound();  
 *assertNotEquals*(prev, post);  
 } catch (IOException | JAXBException | NullPointerException e) {  
 *fail*();  
 }  
 }  
  
 @Test  
 @Order(3)  
 void delete() {  
 try {  
 Session session = *getSession*();  
 JAXBAPI.*executeXML*("files/4.xml", session);  
 PersonEntity p = JAXBAPI.*executeXML*("files/2.xml", session);  
 *assertNull*(p);  
 } catch (IOException | JAXBException e) {  
 *fail*();  
 }  
 }  
  
  
 @Test  
 @Order(4)  
 void toxml() {  
 PersonEntity personEntity = *getSession*().find(PersonEntity.class, "900");  
 PersonXML personXML = new PersonXML(personEntity);  
 try {  
 JAXBContext context = JAXBContext.*newInstance*(PersonXML.class);  
 Marshaller marshaller = context.createMarshaller();  
 marshaller.setProperty(Marshaller.*JAXB\_FORMATTED\_OUTPUT*, Boolean.*TRUE*);  
 BufferedWriter writer = new BufferedWriter(new FileWriter("files/out.xml"));  
 marshaller.marshal(personXML, writer);  
 writer.close();  
 } catch (JAXBException | IOException e) {  
 *fail*();  
 }  
 }  
  
 @Test  
 @Order(5)  
 void jsontest(){  
 try {  
 JAXBAPI.*addFromJson*("files/in.json", *getSession*());  
 JAXBAPI.*dumpToJson*("files/out.json", *getSession*(), "400");  
 JAXBAPI.*executeXML*("files/5.xml", *getSession*());  
 } catch (IOException | JAXBException e) {  
 *fail*();  
 }  
 }  
  
}

Данные два класса могут содержать логику в разном объеме – все это зависит от цели и команды разработчика.